Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Damjan Klemenčič

Aplikacija za analizo podatkov klicnega centra

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Aplikacija za analizo podatkov klicnega centra

Tematika naloge: V diplomski nalogi predstavite načrtovanje, razvoj in uporabo namizne aplikacije za klicne in dispečerske centre, kjer imajo komunikacijski strežnik podjetja Siemens. Namen aplikacije je zbiranje in hramba podatkov komunikacijskega strežnika, na podlagi katerih je mogoče izvajati analize uporabe in generirati različna poročila. V nalogi izpostavite posebnosti povezave med komunikacijskim strežnikom in računalnikom z uporabo protokola CSTA.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Damjan Klemenčič, z vpisno številko **63030218**, sem avtor diplomskega dela z naslovom:

Aplikacija za analizo podatkov klicnega centra

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 18. septembra 2014 Podpis avtorja:

Posebna zavala gre mentorju viš. pred. dr. Igorju Rožancu za prijaznost, njegove nasvete in usmerjanje pri izdelavi diplomske naloge. Prav tako se zahvaljujem ženi Tini in sinu Lanu ter staršem za vso podporo, ki so mi jo nudili med študijem.

Kazalo

Povzetek

Diplomska naloga opisuje načrtovanje in razvoj namizne aplikacije CTI Server, ki je uporabna v klicnih in dipečerskih centrih, kjer imajo komunikacijski strežnik (telefonsko centralo) Siemens. Aplikacija zbira podatke, ki jih pošilja komunikacijski strežnik, in jih shranjuje v podatkovno bazo za kasnejšo obdelavo. Prikazuje tudi trenutno stanje agentov in telefonskih povezav ter zgodovino klicev in odjav agentov za določeno časovno obdobje.
Pri izdelavi aplikacije smo uporabil orodja, kot so `Visual Studio 2008`, `Microsoft SQL Server 2008 R2`. Razvoj je potekal v programskem jeziku `C#`. Glavni del aplikacije je protokol `CSTA`, ki je namenjen komuniciranju s komunikacijskim strežnikom. Povezava je fizično realizirana s `RS232` serijskimi vrati. Razvoj je potekal v štirih korakih, in sicer analiza in načrtovanje, izdelava aplikacije, testiranje v laboratoriju in testiranje v produkciji. Aplikacija lahko s podatki, ki jih pridobi od komunikacijskega strežnika, učinkovito pripomore k organizaciji in nadzoru dela ter oddajanju poročil v klicnem ali dispečerskem centru. Kljub temu, da aplikacija že deluje, ima še vedno nekaj možnosti za nadalnji razvoj.

Ključne besede: komunikacijski server, agent, dispečer, CSTA, klic, aplikacija.

Abstract

Graduation thesis describes the planning and design of desktop application CTI Server that is used in call and dispatch centers, where they have communication server (PBX) Siemens. Application collects data that is sent from communication server and stores them in data base for subsequent processing. It also displays current status of agents and telephony connections, call history and agent logs for a certain period of time.
Tools like `Visual Studio 2008`, `Microsoft SQL Server 2008 R2` was used at the development of application. Development was performed with `C#` programming language. Main part of application is `CSTA` protocol that is intended to communicate with communication server. Connection is physically realized with `RS232` serial port. Development was carried out in four steps namely analysis and planning, application development, testing in laboratory and testing in production. With the data obtained from communication server the application can effectively contributes to the organization and work supervision and reports emission in call or dispatch center. Despite the fact that application is already operational still has some possibilities for further development.

Keywords: communication server, agent, dispatcher, CSTA, call, application.

Poglavje 1

Uvod

Uporaba računalnikov v zadnjih letih močno narašča. S tem narašča tudi potreba po integraciji različnih sistemov z računalniki. Z namenom integracije telefonskega sistema in računalnika je bila razvita aplikacija, ki to omogoča. Ta dva sistema sta bila integrirana s pomočjo protokola CSTA (ang. Computer Supported Telecommunications Applications) [12]. Aplikacija omogoča nadzor celotnega dela klicnega centra (tako telefonov kot osebja) in celo poseganje v delovanje telefonov. Vsak vodja želi biti kar se da dobro informiran o delu svojih podrejenih, kar pa je pri velikem številu zaposlenih težko. Zato ta aplikacija preko komunikacijskega strežnika zbira podatke, ki omogočajo učinkovit nadzor. Tako lahko vodilni učinkovito razporedijo in optimalno izkoristijo delovno silo. Seveda je v klicnih centrih, kjer je telefonska številka plačljiva, zelo pomemben tudi podatek o mesečnem zaslužku. Tudi na to vprašanje si s podatki, ki jih zajame aplikacija, lahko brez težav odgovorimo. V dispečerskih centrih (npr. reševalnih postajah) vodilne zanimajo tudi podatki o tem, koliko časa dispečerju zvoni telefon, preden odgovori, kako dolgo se klici zadržujejo v vrsti in podobno. Ker dobra odzivnost rešuje življenja, je to zelo pomembno. Cilj aplikacije je zajeti vse te in še veliko drugih podatkov ter jih shraniti v podatkovno bazo, kjer so dostopni za nadaljno analizo.

Aplikacija je bila razvita v microsoftovem okolju Visual studio 2008. Napisana je bila v programskem jeziku C# [2]. Za podatkovno bazo smo izbrali

Microsoft SQL Server 2008 R2 [6].

Komunikacijski strežnik, s katerim je povezana aplikacija, je Siemens HiPath3500 [13]. Različica programske opreme je HiPath3000 V1.2 / Hicom 150H [13]. Komunikacija poteka preko RS232 komunikacijskih vrat [3].

V naslednjih poglavjih je predstavljeno, kakšna orodja in tehnologoje smo uporabili, da smo prišli do končnega izgleda in funkcionalnosti aplikacije, kako je potekal razvoj in kakšen je njen izgled ter lastnosti. Na koncu sledijo še sklepne ugotovitve, kjer je opisano, kaj je bilo narejenega in kaj se da še narediti.

Poglavje 2

Uporabljena orodja in tehnologije

Pri razvoju aplikacije smo uporabljali različna orodja in tehnologije, ki so pripeljale do končnega cilja. V nadaljevanju so ta orodja in tehnologije tudi predstavljene.

2.1 Razvojno okolje

Aplikacijo smo razvijali v Microsoft Visual Studio 2008, ki je bil izdan 19. novembra 2007. To je zelo učinkovito orodje za razvoj tako namiznih kot tudi spletnih aplikacij. Je dokaj enostaven za uporabo in učinkovit. Podpira naslednje programske jezike:

- C#
- C++
- Visual Basic

Komponente lahko na grafični vmesnik dodajamo ročno ali programsko. Z uporabo tega razvojnega orodja v kombinaciji s programskim jezikom C# [2] imam zelo dobro izkušnje. Okolje nudi veliko pomoči pri postavitvi kontrolnikov na formo, saj pri postavitvi kontrolnik na formo vso kodo zgenerira sam

(s pomočjo tega lahko privarčujemo na času), lahko pa vso kodo napiše tudi uporabnik. Okolje je zelo intuitivno in zaradi tega tudi uporabniku prijazno [1].

2.2 Podatkovna baza

Za shranjevanje podatkov smo uporabili podatkovno bazo Microsoft SQL Server 2008 R2. To je verzija 10.50.1600.1, prej znana pod razvojnim imenom ″Kilimanjaro″, ki je bila najavljena na TechEd 2009. Uporabniki so jo lahko začeli uporabljati 21. aprila 2010. Verzija R2 doda določene funkcionalnosti verziji SQL Server 2008, med njimi tudi sistem za splošno upravljanje podatkov (ang. master data manedgement - MDM), ki ga Microsoft uporablja pod blagovno znamko Master Data Services. Ena teh funkcij je tudi centralizirana konzola za upravljanje več SQL Server 2008 instanc in servisov z imenom Multi Server Management.
Podatkovna baza omogoča veliko funkcionalnosti, zato nismo imeli nikakršnih težav z namestitvijo baze v našem okolju. Tudi z uporabo SQL strežnika nismo imeli nikakršnih težav, saj je dokaj enostaven za uporabo [6].

2.3 Povezava komunikacijskega strežnika in računalnika

Komunikacijski strežnik in računalnik sta povezana z RS-232 kablom [3]. Standard je bil izdan leta 1969 s strani Standards Committee, danes znane kot Electronic Industries Association. Takrat se je uporabljal za naprave za prenos podatkov. Prenos podatkov je v tistem času pomenil digitalno izmenjavo podatkov med računalnikom in oddaljenim terminalom ali med dvema terminaloma. Povezava je bila vzpostavljena tako, da so bili terminali priklopljeni preko telefonskih linij, zato so bili potrebni modemi na vsakem koncu linije, da so skrbeli za prenos podatkov. Brez teh modemov je imel digitalni prenos podatkov preko analognega kanala svoje napake. Tako je

nastal nov standard, ki zagotavlja zanesljivo povezavo, hkrati pa je poskrbel, da so različni proizvajalci komunikacijske opreme brez težav poskrbeli za združljivost naprav drugih proizvajalcev.
Zahteve tega standarda določajo stopnjo signala (izhodni signal ponavadi niha od +12V do -12V), čas signala, funkcijo signala, vtiče in protokol za izmenjavo podatkov. V več kot 30 letih je bil standard trikrat spremenjen, nazadnje leta 1991, ko je bil preimenovan v EIA/TIA-232-E. Še vedno je v splošni rabi ime RS-232-C ali samo RS-232 [3].
Standard določa dve vrsti priključkov:

- DB-25 pinski vtič
- DB-9 pinski vtič

Vsak od teh priključkov je lahko moški (ang. Data Terminal Equipment - DTE) ali ženski (ang. Data Comunnication Equipment - DCE). DB-9 pinski vtič je prikazan na sliki 2.1, razpored pinov pa v tabeli 2.1. DB-25 pinski priključek ni posebaj prikazan, ker ga pri tem projektu nismo uporabljali. Aplikacija za branje podatkov iz komunikacijskega strežnika uporabilja 9 pinski RS232 priključek [4].

Slika 2.1: 9 pinski RS232 priključek

Pin	Ime signala	Opis
1	DCD	Podatkovni nosilec zaznan
2	RxD	Prejeti podatki
3	TxD	Oddani podatki
4	DTR	Podatkovni terminal pripravljen
5	GND	Ozemljitev
6	DSR	Podatkovni paket pripravljen
7	RTS	Zahteva za pošiljanje
8	CTS	Možno pošiljanje
9	RI	Zaznavanje obroča

Tabela 2.1: Razpored pinov pri 9 pinski RS232 priključku

2.4 Protokol CSTA

Povezava do računalnika je definirana po protokolu CSTA (ang. Computer Supported Telecomunications Applications), ki je definiran v standardih ECMA-217 [10] in ECMA-218 [11]. ECMA (ang. European Computer Manufacturers Association) je bila ustanovljena leta 1961. Leta 1994 je bilo ime spremenjeno v Ecma International - European association for standardizing information and communication systems. Vendar se je blagovna znamka ″Ecma″ obdržala zaradi zgodovinskih razlogov. V njej so sodelovali strokovnjaki na področjih telefonskih central, računalniške opreme in predstavniki telekomunikacijskih podjetij. Vsa ta področja so pomembna za CTI (Computer Telephone Integration) [7]. Najprej so raziskave temeljile predvsem na tem, kaj potencialni kupci pravzaprav pričakujejo od CTI aplikacij in tipičnih primerih uporabe. Poizkušali so zgraditi protokol, ki je povsem neodvisen od nižjih transportnih slojev, zato je na nivoju CSTA povsem nepomembno, s pomočjo kakšne povezave poteka komunikacija. Pomembno je le, da povezava obstaja [9]. Protokol omogoče tudi klicanje, odgovarjanje na klice, itd. Vsi dogodki in stanja, ki so zajeti v poglavju 4.3, so definirani v ECMA CSTA protokolu [10].

Vse CSTA aplikacije so zasnovane na OSI ISO 7 slojnem modelu in upora-

bljajo sloje 1, 2, 5 in 7 [12]. Model je podan v tabeli 2.2.

Vmesnik	RS-232
7. sloj	aplikacijski sloj (CSTA)
6. sloj	ASN.1
5. sloj	sejni sloj
4. sloj	neuporabljen
3. sloj	neuporabljen
2. sloj	podatkovni sloj (konferenčni)
1. sloj	fizični sloj (serijski port)

Tabela 2.2: OSI ISO model

2.4.1 Predstavitev CSTA povezave za Siemens

Za testiranje smo uporabili komunikacijski strežnik HiPath 3500 z verzijo programske opreme HiPath 3000 V1.2/Hicom 150H [13]. S komunikacijskim strežnikom se lahko povežemo na tri načine:

- povezava preko RS-232 (aplikacija uporablja to vrsto povezave),
- povezava preko ISDN,
- povezava preko TCP/IP.

Napravi smo med seboj povezali s kablom (kabel ne sme biti daljši od 15 metrov). Nastavitve, ki smo jih v aplikaciji uporabili za komunikacijo preko serijskega porta, so:

- hitrost prenosa: 19200,
- podatkovni biti: 8,
- paritetni biti: None,
- zaključni biti: 1.

Pred prenosom podatkov znotraj aplikacije CSTA morajo biti vzpostavljeni sloji 2, 5 in 7. Na 2. sloju (podatkovni sloj) se uporablja BSCSUB [12] protokol, ki je izpeljan iz BSC [5] protokola in podpira naslednje funkcije:

- polni dvostranski prenos,
- zaprt prenos z uporabo `STX/ETX` znakov,
- transparentnost z uporabo `DLE` kot ubežnega znaka,
- detekcijo napak pri prenosu z uporabo kontrolne vsote,
- ponovitev pokvarjenega podatkovnega bloka z uporabo `NAK` in `ENQ` (ang. enquiry)
- potrjevanje prenosa z uporabo `ACK0/ACK1` (s tem se izognemo izgubi podatkovnih blokov ali dvojnemu pošiljanju le-teh).

Vrednost poslanega bajta je predstavljena kot heksadecimalno število v sintaksi programskega jezika C (`0x00, 0x01, 0x02` ...). Vsi podatki se prenašajo v podatkovnih blokih. Primer podatkovnega bloka je prikazan v tabeli 2.3.

DLE	STX	podatki za 5. sloj	DLE	ETX	LRC
0x10	0x02	podatki za 5. sloj	0x10	0x03	LRC

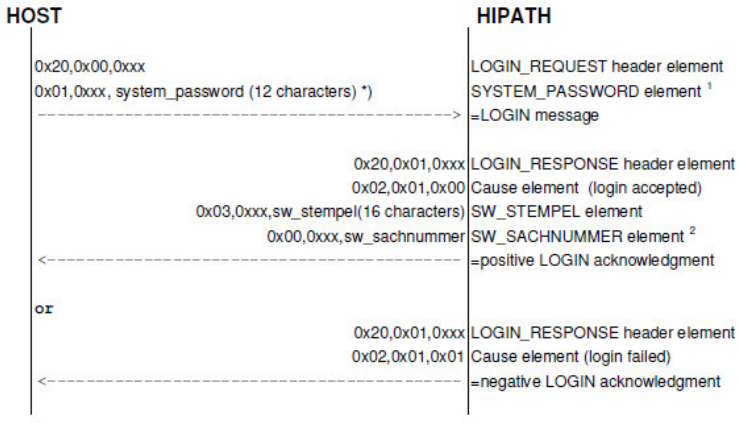
Tabela 2.3: Format podatkovnega bloka

V bloku lahko manjkajo podatki za 5. sloj, kar imenujemo prazen blok. Ko aplikacija sprejme tak blok, najprej preveri, ali je ustrezen in ga potrdi z `ACK0` (`0x10, 0x30`) ali z `ACK1` (`0x10, 0x31`). Če se kontrolna vsota ne ujema, blok ni ustrezen in aplikacija vrne negativno potrditev `NAK` (`0x15`). Če ni potrditve, se pošlje poizvedovanje (`DLE, ENQ`).
Vzpostavitev OSI sloja 2 se lahko začne iz HiPath ali strežniške strani. To se naredi tako, da se pošlje prazen blok in se čaka na potrditev. Če ni potrditve, se lahko čez 20 sekund pošlje nov prazen blok, kar se lahko zgodi trikrat in

vodi do 60 sekundne zamude. Po preteku tega časa se ugotovi, da povezave ni možno vzpostaviti.
Po vzpostavitvi podatkovnega sloja se lahko vzpostavi OSI sloj 5 (sejni sloj). Ta sloj mora biti vzpostavljen preden začnemo pošiljati podatke. Sporočilo za 5. sloj je sestavljeno iz glave in podatkov za višji sloj. Glava seje ima naslednjo strukturo:

- sistemska aplikacija (v našem primeru `CTI_APPLICATION 0x20`),
- storitev seje (odpri sejo, zapri sejo ...),
- dolžina sporočila.

Odprtje seje lahko začne samo strežnik. Potek je prikazan na sliki 2.2.



Slika 2.2: Primer odpiranja seje

Slika 2.2 prikazuje primer odpiranja seje. Strežnik (gostitelj) pošlje komunikacijskemu strežniku zahtevo, ki vsebuje `LOGIN_REQUEST` in `SYSTEM_pASSWORD`. Točna sestava sporočila za vzpostavitev seje se nahaja v tabeli 2.4.
Potem komunikacijski strežnik pošlje odgovor z bodisi pozitivno ali negativno potrditvijo. Če je potrditev pozitivna, dobimo poleg `LOGIN_RESPONSE` in vzroka (prijava sprejeta) tudi `SW_STEMPEL` (verzija programske opreme proizvajalca) in `SW_SACHNUMMER` (tovarniška verzija programske opreme), drugače pa samo `LOGIN_RESPONSE` in vzrok (prijava ni uspela) [12].

DLE	0x10
STX	0x02
CTI_APPLICATION	0x20
LOGIN_REQUEST	0x00
LENGTH	0x07
SYSTEM_PASSWORD	0x01
LENGTH	0x05
PASSWORD	0x03 0x01 0x09 0x09 0x04
DLE	0x10
ETX	0x03
LRC	0x5A

Tabela 2.4: Zahteva za odpiranje seje

Ko je povezava na 5. sloju vzpostavljena, se lahko začne prenos podatkov na 7. aplikacijskem sloju. Ta sloj je definiran v standardu ECMA-218 [11] z nekaterimi razlikami, saj so znane napake tukaj že odpravljene. Odpiranje seje je zadji korak, preden komunikacijski strežnik začne konstantno pošiljati poročila o dogajanju o delu agentov in dogajanju na nadzorovanih napravah.

Poglavje 3

Razvoj aplikacije

Razviti smo želeli aplikacijo, ki bi lahko omogočala celovit nadzor nad delom v klicnem centru, izračun prihodkov na podlagi prejetih klicev in minimiziranje stroškov z delovno silo s pravilno razporeditvijo le-te. Razvoj aplikacije je potekal v treh korakih:

- analiza zahtev in načrtovanje,
- izdelava aplikacije,
- testiranje v laboratoriju ter
- testiranje v produkciji.

V nadaljevanju so ti koraki opisani.

3.1 Analiza zahtev in načrtovanje

Pred začetkom projekta je bilo potrebno preučiti vso dokumentacijo in ugotoviti, kaj bi aplikacija lahko ponujala. Potem smo se dogovorili, kako naj bi aplikacija izgledala in kakšne funkcionalnosti naj bi imela, pri čemer so bile v ospredju želje strank. Pri izdelavi načrta smo bili omejeni s tehničnimi možnostmi komunikacijskega strežnika, vendar to ni predstavljalo bistvenih težav, saj ni bilo zahtevano nič, kar se tehničnim omejitvam navkljub ne

bi dalo implementirati. Zahtevano je bilo, da aplikacija zbira podatke, s pomočjo katerih se lahko naredi določena poročila, za učinkovitost delovanja klicnega ali dispečerskega centra. Glede agentov so pomembna predvsem vprašanja:

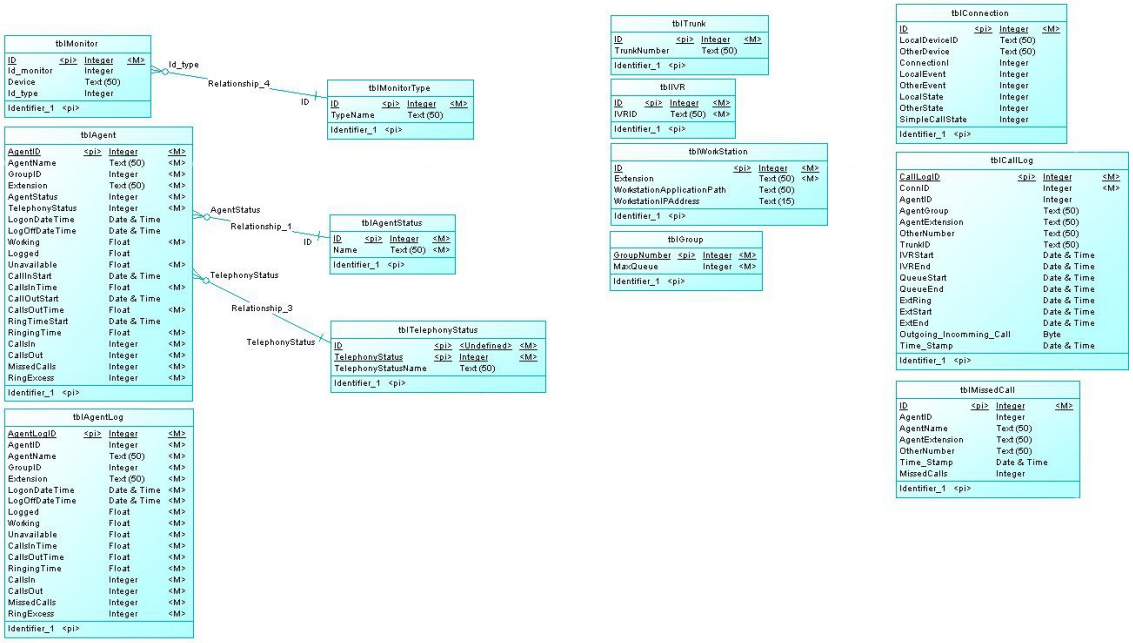
- koliko je neodgovorjenih klicev,
- koliko je klicev,
- koliko je klicev, katerih zvonjenje traja dlje od 10 sekund,
- koliko dohodnih klicev je sprejel agent,
- koliko odhodnih klicev je opravil (s pomočjo tega podatka lahko izračunamo, kakšne stroške je povzročil),
- kakšno je skupno trajanje dohodnih in odhodnih klicev,
- koliko časa mu zvoni telefon, preden se oglasi,
- koliko časa je nerazpoložljiv in
- koliko časa je imel status delo.

Tudi glede klicev je nekaj pomembnih vprašanj, na katera v klicnih centrih potrebujejo odgovore. Ta vprašanja so sledeča:

- kdaj je klic prišel v vrsto in kdaj je vrsto zapustil (na teh dveh podatkov lahko izračunamo, koliko časa povprečno stranke čakajo v vrsti),
- kdaj je agentu začel zvoniti telefon,
- kdaj se je odzval na klic,
- kdaj je končal klic in ali je klic odhoden ali dohoden (s pomočjo tega podatka lahko izračunamo prihodke, če je telefonska številka plačljiva, in stroške).

Aplikacija zbira odgovore na vsa ta vprašanja. Ko dobimo odgovore na vsa zastavljena vprašanja, lahko učinkoviteje načrtujemo delo v dispečerskem ali klicnem centru.

Ko smo naredili natančen načrt aplikacije, smo se lotili izdelave načrta podatkovne baze. Poimenovali smo jo `Cti` in vsebuje trinajst tabel, v katerih se hranijo podatki, pomembni za razvoj aplikacije. Slika 3.1 prikazuje podatkovni model. V tabeli `tblMonitor` se nahajajo podatki o nadzorovanih napravah. Nadzorovane naprave se prepišejo iz tabel `tblTrunk, tblIVR, tblWorkStation in tblGroup`. V teh tabelah se nahajajo vsi možni tipi naprav, ki so pomembni za delovanje klicnega centra. Tabela `tblMonitor` je v relaciji s tabelo `tblMonitorType`, kjer se nahaja identifikacijska številka tipa ter ime naprave. Imena naprav so opisana v poglavju 4.5. Tabela `tblAgent` je tabela, ki je namenjena prikazovanju trenutnega stanja agentov in trenutnega stanja njihovih telefonov. Ta tabela je v relaciji s tabelama `tblAgentStatus` in `tblTelephonyStatus`. V prvi tabeli se nahajajo identifikacijske številke agentskih stanj in njihovo ime, v drugi pa identifikacijske številke telefonskih stanj in njihova imena. Ta stanja so opisana v poglavju 4.2.2. Tabela `tblAgentLog` je namenjena prikazu zgodovine agenta. Vsi podatki, razen `LogOffDateTime`, ki se izračuna sproti, se v to tabelo prepišejo iz table `tblAgent`, ko se agent odjavi. V tabelo `tblConnection` se vpisujejo trenutne telefonske povezave in njihovo stanje. Nova vrstica se naredi, ko naprava postane aktivna, zbriše pa se, ko naprava postane neaktivna. Če so vse naprave neaktivne, v tej tabeli ni nobenega zapisa. Tabela `tblCallLog` je namenjena zgodovini klicev. Vanjo se shrani vsak klic od začetka (npr.: ko pride klic v vrsto) do konca, bodisi da se agent nanj odzove ali ne. V tabelo `tblMissedCall` se shranjujejo podatki o neodgovorjenih klicih. Ko se noben agent ne odzove na klic, se v tej tabeli naredi nova vrstica. Ko je podatkovna baza zgrajena, lahko pričnemo z razvojem aplikacije.



Slika 3.1: Table v bazi `Cti`

3.2 Izdelava aplikacije

Najprej smo razvili kontrolnik `ctlCC`, ki vsebuje štirideset razredov, s pomočjo katerih obdeluje sporočila, ki jih pošilja komunikacijski strežnik. Nekaj razredov pripravi ukaze za komunikacijski strežnik:

- `cOSILayer2`, za vzpostavitev podatkovne povezave,
- `cOSILAyer5`, za vzpostavitev seje
- `cColMonitor`, za pričetek nadzorovanja naprave

Kontrolnik je namenjen komunikaciji s komunikacijskim strežnikom in obdelavi sporočil. Prvi korak razvoja je bil povezava na 1. sloju, drugi korak povezava na 2. sloju in tretji korak povezava na 5. sloju OSI modela. `cColMonitor` je razred, ki preko dogodka `eChangeMonitor` pošilja podatke v aplikacijo. Le-ta vsebuje tudi zbirko vseh nadzorovanih naprav.

V nadaljevanju je predstavljen primer dogodka `ConnectionCleared` iz funkcije `CallEvent` v razredu `cColMonitor`.

```
1  case cCallEvent.eCallEventType.ConnectionCleared:
2          this._ctlCC.Log("KONTROLNIK Call E., Connection
               Cleared E., MonitorID/DeviceID/ConnectionID " +
               cRose.Invoke.Argument.MonitorCrossRefID + "/" +
                       cRose.Invoke.Argument.EventSpecInfo.
               CallEvent.ConnectionCleared.DroppedConnection.
               DeviceID + "/" + cRose.Invoke.Argument.
               EventSpecInfo.CallEvent.ConnectionCleared.
               DroppedConnection.CallID);
3          this._ctlCC.Log("\tDropped Connection ConnnnectionID:
                " + cRose.Invoke.Argument.EventSpecInfo.CallEvent
               .ConnectionCleared.DroppedConnection.CallID);
4          this._ctlCC.Log("\tDropped Connection DeviceID: " +
               cRose.Invoke.Argument.EventSpecInfo.CallEvent.
               ConnectionCleared.DroppedConnection.DeviceID);
5          this._ctlCC.Log("\tReleasing Device: " + cRose.Invoke
               .Argument.EventSpecInfo.CallEvent.
               ConnectionCleared.ReleasingDevice);
6  //-------------------------------------------------------
7
8          ushortConnectionID = cRose.Invoke.Argument.
               EventSpecInfo.CallEvent.ConnectionCleared.
               DroppedConnection.CallID;
9          strReportedDeviceID = cRose.Invoke.Argument.
               EventSpecInfo.CallEvent.ConnectionCleared.
               DroppedConnection.DeviceID;
10
11         if (this.MonitorCrossRefIDExists(cRose.Invoke.
               Argument.MonitorCrossRefID))
12         {
13                 if (this.GetMonitorDevice(cRose.Invoke.
                       Argument.MonitorCrossRefID).DeviceID.
                       Equals(strReportedDeviceID))
14                 {
15                         //local device
```

```
16                          if (this._monitorCol[
                               strReportedDeviceID].ColConnection
                               .ContainsConnectionID(
                               ushortConnectionID))
17                          {
18                                  strLocalDeviceID =
                                       strReportedDeviceID;
19                                  this._ctlCC.Log("KONTROLNIK
                                       -> Connection Cleared:
                                       Remove Connection --> this
                                       ._monitorCol[" +
                                       strLocalDeviceID + "].
                                       ColConnection.Remove(" +
                                       ushortConnectionID.
                                       ToString() + ")");
20                                  this._monitorCol[
                                       strLocalDeviceID].
                                       ColConnection.Remove(
                                       ushortConnectionID);
21                                  intLocalEvent = (int)cRose.
                                       Invoke.Argument.
                                       EventSpecInfo.CallEvent.
                                       CallEventType;
22                                  intLocalState = (int)
                                       ConnectionState.StateNull;
23                          }
24                  }
25                  else
26                  {
27                          //other device
28                          strLocalDeviceID = this.
                               GetMonitorDevice(cRose.Invoke.
                               Argument.MonitorCrossRefID).
                               DeviceID;
29                          strOtherDeviceID =
                               strReportedDeviceID;
30                          intOtherEvent = (int)cRose.Invoke.
                               Argument.EventSpecInfo.CallEvent.
```

```
                        CallEventType;
31                  intOtherState = (int)ConnectionState.
                        StateNull;
32              }
33
34              strTimeStamp = cRose.Invoke.Argument.
                    CSTACommonArguments;
35              this.fireEvtChangeMonitor(strLocalDeviceID,
                    strOtherDeviceID, ushortConnectionID,
                    intLocalEvent, intOtherEvent,
                    intLocalState, intOtherState, cRose.Invoke
                    .Argument.MonitorCrossRefID, intAgentEvent
                    , intAgentState, intAgentID, strTimeStamp)
                    ;
36          }
37          break;
```

Zgornja koda obdeluje primer, ko se zgodi dogodek `ConnectionCleared`. Prvi del kode do vrstice osem zapiše v dnevnik, kateri dogodek se je zgodil in nekaj parametrov (na katerem monitorju se je dogodek zgodil, katera naprava pripada temu monitorju, številko povezave in še nekaj drugih), ki v primeru težav pomagajo odkriti, kje je prišlo do napake. Potem v spremenljivki `ushortConnectionID in strReportedDeviceID` shranimo številko povezave in številko naprave (oba podatka sta del sporočila, ki ga pošlje komunikacijski strežnik). Sledi preverjanje, če obstaja referenca na to napravo. To se preveri s klicom funkcije `MonitorCrossRefIDExists` s parametrom `cRose.Invoke.Argument.MonitorCrossRefID` (tudi parameter je del sporočila, ki ga pošlje komunikacijski strežnik). Nato s funkcijo `GetMonitorDevice` in s prejšnjim parametrom dobimo številko naprave. Če je številka naprave enaka `strReportedDeviceID`, pomeni, da je sporočilo o monitorirani napravi, drugače pa je sporočilo o drugi napravi. V primeru, da je sporočilo o monitorirani napravi, preverimo, če na tej napravi obstaja številka povezave `ushortConnectionID`. Če obstaja, se napolnijo naslednje spemenljivke:

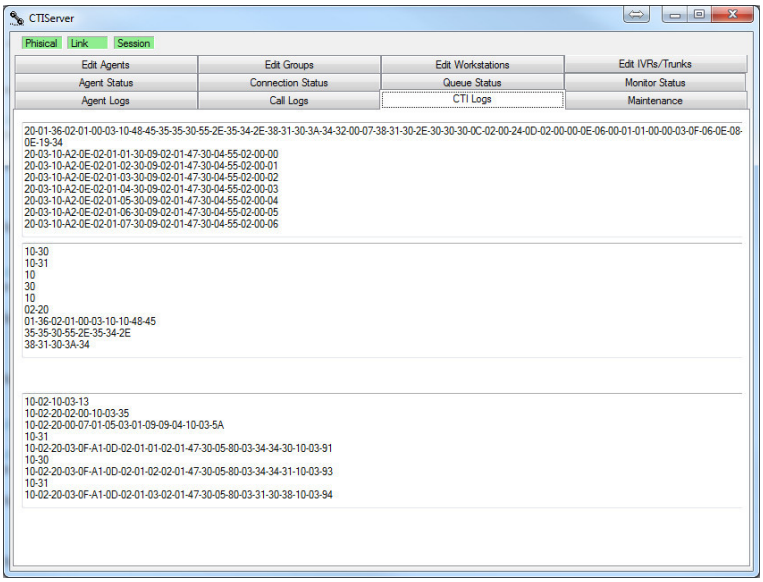
- `strLocalDeviceID`, v katero se zapiše številka naprave za katero je sporočilo (`strReportedDeviceID`),
- `intLocalEvent`, v katero se zapiše številka dogodka, ki se je zgodil (glej poglavje 4.3.1),
- `intLocalState`, v katero se zapiše številka novega stanja povezave (glej poglavje 4.3.1).

V vrstici devetnajst se v dnevnik zapiše številka povezave, ki bo odstranjena iz zbirke, v vrstici dvajset pa se ta številka povezave odstrani iz zbirke.
Če je sporočilo o drugi napravi, se izvajanje kode nadaljuje od vrstice petindvajset do dvaintrideset. V tem primeru se napolnijo naslednje spremenljivke:

- `strLocalDeviceID`, v katero se zapiše številka naprave, ki je povezana s poročano referenčno številko (`this.GetMonitorDevice(cRose.Invoke.Argument.MonitorCrossRefID).DeviceID`) in to je tudi naprava, ki je nadzorovana na tej referenčni številki,
- `strOtherDeviceID`, v katero se zapiše številka naprave, kateri je namenjeno sporočilo (`strReportedDeviceID`),
- `intOtherEvent`, v katero se zapiše dogodek, ki se je zgodil na drugi napravi,
- `intOtherState`, v katero se zapiše trenutno stanje druge naprave.

V vrstici štiriintrideset se napolni spemenljivka `strTimeStamp` (časovni žig - komunikacijski strežnik ga pošlje skupaj s sporočilom o tem dogodku) in na koncu v vrstici petintrideset se sproži še dogodek, ki vse te podatke pošlje v glavno formo.

Ko je bil kontrolnik pripravljen, smo ga postavili na testno okolje, da smo preizkusili njegovo delovanje. Tedaj smo začeli razvijati uporabniški vmesnik. Na koncu pa smo vse to združili v eno aplikacijo. Slika 3.2 prikazuje postavitev kontrolnika `ctlCC` na zavihek ″CTI Logs″.



Slika 3.2: Kontrolnik ctlCC

3.3 Testiranje v laboratoriju

Ko je bila razvita celotna aplikacija, smo lahko pričeli s testiranjem. Testirali smo različne scenarije (npr. odhodni klic, dohodni klic, prijava agenta, odjava agenta, itd.) in preverjali, ali dobimo pravi odziv ali ne. Med testiranjem so se pojavljale različne napake, ki smo jih sproti odpravljali.

Najbolj pogoste napake so bile, da so se nekateri klici narobe beležili ali le-ti niso bili združeni v eni vrstici od začetka do konca (ko je prišel klic v vrsto, je nastala nova vrstica, ko je prispel do agenta, je zopet nastala nova vrstica). Zgodilo se je tudi, da je aplikacija obstala, če je bila povezava nekaj časa neaktivna (to se je zgodilo zato, ker je komunikacijski strežnik prekinil povezavo zaradi neaktivnosti).

3.4 Testiranje v produkciji

Ko je aplikacija v laboratoriju stabilno delovala, je bila pripravljena na testiranje v realnem okolju. Postavili smo jo v klicni center v Celju. Pri testiranju

pa smo občasno naleteli na težave, ki smo jih lahko s pomočjo dnevnika odkrili. Največkrat je šlo za odsotnost kakšnega od sporočil. V uporabniškem vmesniku se je to najbolje videlo v zavihku ″Call Logs″, kjer je kakšen del klica preprosto manjkal. Recimo, ko je agentu začelo zvoniti, se je v koloni ″ExtRing″ zabeležil čas zvonenja, ko se je javil, se je v koloni ″ExtStart″ zabeležil čas začetka pogovora, ko pa se je klic končal, se v koloni ″ExtEnd″ ni zabeležilo nič. Ko je agent dobil nov klic, so se kolone polnile po enakem postopku in tako je izgledalo, da se je začela nova povezava, preden se je stara končala. Zato je bilo potrebno v kodi popravljati napake komunikacijskega strežnika s tako imenovanimi samozdravilnimi mehanizmi. Eden od teh mehanizmov se nahaja v razredu `cColMonitor` v funkciji `CallEvent` v primeru dogodka (ang. case) `cCallEvent.eCallEventType.ServiceInitiated`. Če pride do tega dogodka in če ima naprava, za katero je prišlo sporočilo, že kakšno povezavo, ki ni v stanju `ConnectionState.Hold`, je potrebno najprej to povezavo odstraniti, tako da simuliramo dogodek `ConnectionCleared`. Izsek kode za opisan primer je prikazan spodaj.

```
1  //SelfHealing ------------------------------------------------
2  if (this._monitorCol.ContainsKey(strLocalDeviceID) && this.
       _monitorCol[strLocalDeviceID].ColConnection.
       ConnectionCount > 0)
3  {
4          for (int i = 0; i < this._monitorCol[strLocalDeviceID
               ].ColConnection.ConnectionCount; i++)
5          {
6                  if (this._monitorCol[strLocalDeviceID].
                       ColConnection.ConnectionIDAtIndex(i) !=
                       ushortConnectionID && this._monitorCol[
                       strLocalDeviceID].State != (int)
                       ConnectionState.Hold)
7                  {
8                          strLocalDeviceID = cRose.Invoke.
                               Argument.EventSpecInfo.CallEvent.
                               ServiceInitaiated.ConnectionID.
                               DeviceID;
```

```
9                       intLocalEvent = (int)cCallEvent.
                            eCallEventType.ConnectionCleared;
10                      intLocalState = (int)ConnectionState.
                            StateNull;
11                      this._ctlCC.Log("KONTROLNIK ->
                            SelfHealing Issue: Service
                            Initiated E. Remove Connection -->
                             this._monitorCol[" +
                            strLocalDeviceID + "].
                            ColConnection.Remove(this.
                            _monitorCol[" + strLocalDeviceID +
                             "].ColConnection.
                            ConnectionIDAtIndex(" + i + "))="
                            + this._monitorCol[
                            strLocalDeviceID].ColConnection.
                            ConnectionIDAtIndex(i));
12                      this._monitorCol[strLocalDeviceID].
                            ColConnection.Remove(this.
                            _monitorCol[strLocalDeviceID].
                            ColConnection.ConnectionIDAtIndex(
                            i));
13                      this._ctlCC.Log("KONTROLNIK ->
                            SelfHealing Issue: Service
                            Initiated E. this._monitorCol[" +
                            strLocalDeviceID + "].
                            ColConnection.ConnectionCount=" +
                            this._monitorCol[strLocalDeviceID
                            ].ColConnection.ConnectionCount);
14                      strTimeStamp = cRose.Invoke.Argument.
                            CSTACommonArguments;
15                      this.fireEvtChangeMonitor(
                            strLocalDeviceID, strOtherDeviceID
                            , ushortConnectionID,
                            intLocalEvent, intOtherEvent,
                            intLocalState, intOtherState,
                            cRose.Invoke.Argument.
                            MonitorCrossRefID, intAgentEvent,
                            intAgentState, intAgentID,
```

```
                                strTimeStamp);
16                  }
17          }
18  }
19  //End SelfHealing-----------------------------------------
```

Občasno je aplikacija od komunikacijskega strežnika dobila tudi nerazumljivo sporočilo, ki ga ni znala predelati in je zaradi tega prišlo do napake. Tudi take težave smo odpravljali s prirejanjem kode.

V tem poglavju smo opisali, kako je potekal razvoj aplikacije. Zajeli smo vse od analize do testiranja v produkciji. Glavni del je bil izdelava kontrolnika `ctlCC`. Za približno predstavo, kako kontrolnik odreagira na pročila o določenih dogodkih, je bil objavljen del kode za primer dogodka `ConnectionCleared`. Drugi del kode prikazuje samozdravilni mehanizem v primeru odsotnosti kakšnega sporočila.

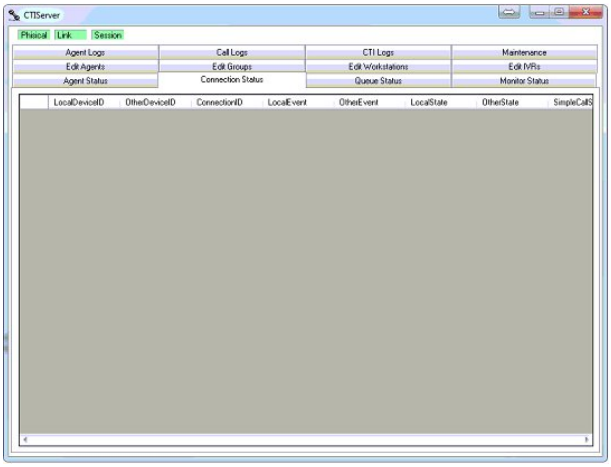
Poglavje 4

O aplikaciji

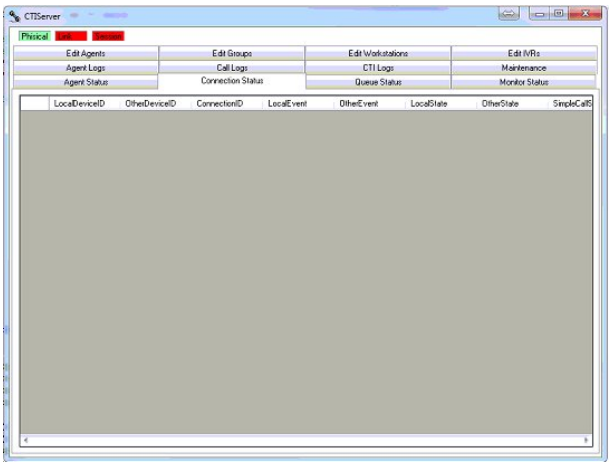
Namen aplikacije je branje in shranjevanje podatkov, ki jih dobimo iz komunikacijskega strežnika. Ti podatki so namenjeni vodjem klicnega centra ali vodjem dispečerjev, da si pridobijo informacije, ki jih zanimajo, in tako kar najbolj optimizirajo delo v klicnem centru.

4.1 Uporabniški vmesnik

Uporabniški vmesnik je zelo preprost. Imamo eno formo, na katero je postavljenih dvanajst zavihkov. Zavihki so razdeljeni v tri vrste po štiri zavihke. Štirje zavihki so namenjeni urejanju, naslednji štirje prikazu trenutnega stanja, trije za pregled zgodovine in eden vzdrževanju. Na vrhu forme so še tri kontrolne oznake, ki kažejo uspešnost povezave s komunikacijskim strežnikom na posameznih nivojih. V večini posameznih zavihkov se nahaja podatkovna mreža, v kateri se prikažejo podatki. Na sliki 4.1 je prikazan izgled, ko se aplikacija pravilno poveže na vseh nihojih, na sliki 4.2 pa izgled vmesnika, ko aplikacija ni mogla vzpostaviti povezave na podatkovnem nivoju ter posledično tudi ne na vseh višjih nivojih.



Slika 4.1: Izgled programa CTIServer



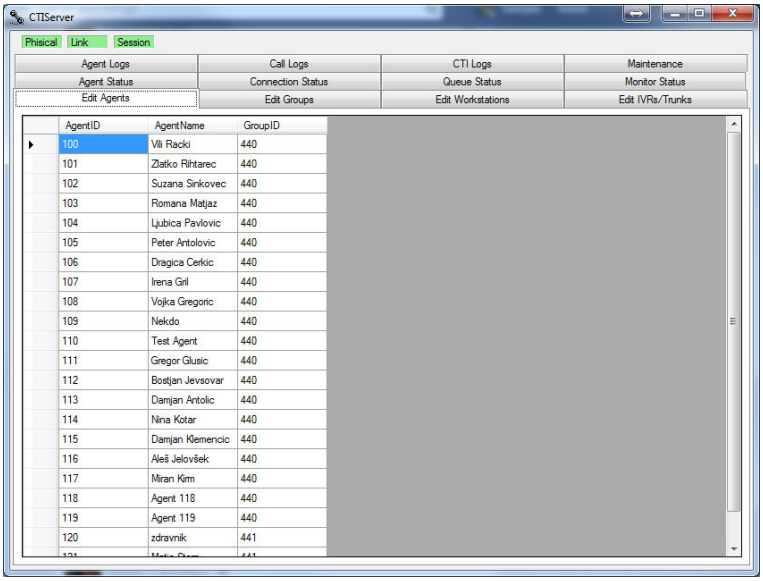
Slika 4.2: CTIServer brez RS-232 povezave

4.2 Agenti

Agenti so uslužbenci klicnega centra, ki delajo na določenem delovnem mestu, tega pa predstavljata telefon in računalnik. Agentom so v programu namenjeni trije zavihki, in sicer ″Edit Agents″, ″Agent Status″ in ″Agent Logs″.

4.2.1 Urejanje agentov

Zavihek ″Edit Agents″ je namenjen urejanju agentov. Agenta lahko dodamo, brišemo ali pa samo spremenimo njegove podatke. Agenti so prikazani v podatkovni mreži. Naslovne kolone so ″AgentID″, ″AgentName″ in ″AgentGroup″. ″AgentID″ predstavlja identifikacijsko številko agenta. S to številko se tudi prijavi v telefon. Ko se prijavi, začne dobivati klice strank, ki jim mora streči z informacijami. V koloni ″AgentName″ sta zapisana ime in priimek agneta, kolona ″AgentGroup″ pa predstavlja skupino, v katero se agent lahko prijavi. Skupin je lahko več in vsaka skupina lahko streže z različnimi informacijami, npr. skupina 440 je namenjena za tehnične težave, skupina 441 za splošne informacije, skupina 442 za račune, ipd. Zavihek za urejanje agentov je prikazan na sliki 4.3.



Slika 4.3: Urenjanje agentov

4.2.2 Trenutno stanje agentov

Zavihek ″AgentStatus″ je namenjen prikazu trenutnih stanj agentov in služi tekočemu spremljanju dogajanja v klicnem ali dispečerskem centru. V koloni

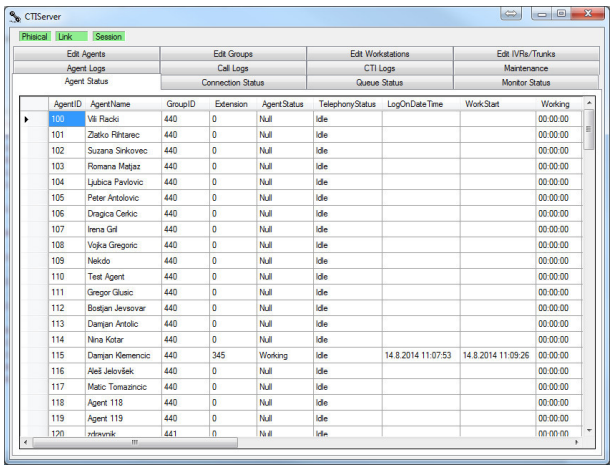
″Extension″ vidimo, na kateri telefon oziroma delovno mesto se je agent prijavil, kolona ″AgentStatus″ pa nam pove, kakšno je trenutno stanje agenta. Stanja so lahko naslednja:

- `Null` - agent je neaktiven
- `Ready` - agent je razpoložljiv
- `Busy` - agent je zaseden
- `Working` - agent ima delo (ta status pride v poštev, če mora agent po pogovoru s stranko izpolniti še obrazec ali v kakšno aplikacijo vnesti podatke, ki jih je pridobil od stranke)
- `Unavailable` - agent je nedosegljiv

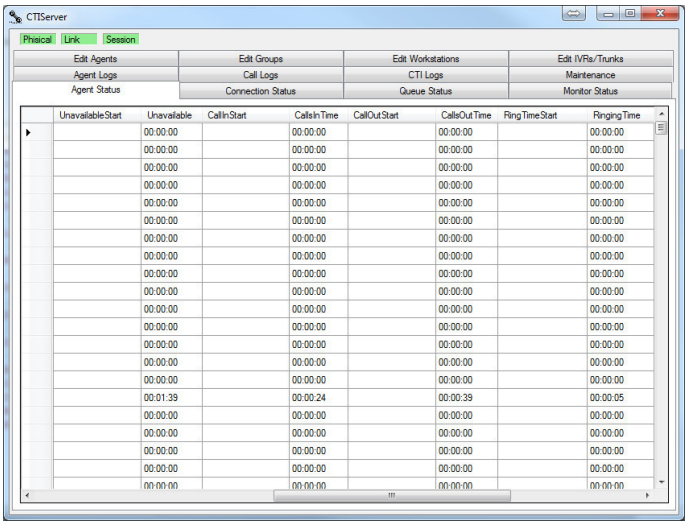
Stanji `Ready` in `Busy` sta avtomatski in agent ne more vpljivati nanju. Stanje `Working` je lahko nastavljeno avtomatsko po vsakem klicu (če je tako določeno v komunikacijskem strežniku), lahko pa ga agent izbere sam s pritiskom na gumb na telefonu. Stanje `Unavailable` je povsem odvisno od agenta in ga prav tako izbere s pritiskom na gumb na telefonu.
Kolona ″TelephonyStatus″ prikazuje trenutno telefonsko stanje agenta. Telefonska stanja so naslednja:

- `Idle` - telefon je v mirovanju
- `ServiceRequest` - agent dvigne telefonsko slušalko
- `Ringing` - agent naredi klic iz svojega telefona in nasprotni strani zvoni
- `Ringed` - zvoni agentov telefon
- `Connected` - agentov telefon in nasprotna stran sta povezana
- `Held` - klic je na čakanju
- `Failed` - telefonska zveza se ni uspela vzpostaviti

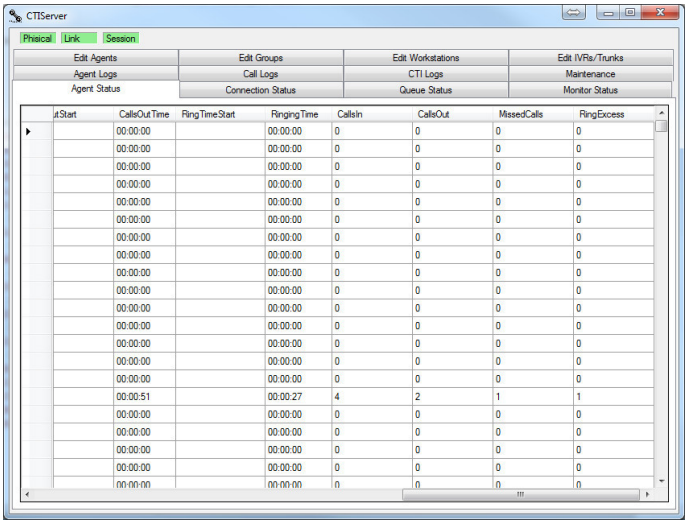
Kolona ″LogOnDateTime″ nam pove čas, ko se je agent s telefonom prijavil v sistem. V kolonah ″WorkStart″ in ″UnavailableStart″ sta prikazana datum in čas agentove izbire enega izmed dveh stanj. Koloni ″Working″ in ″Unavailable″ prikazujeta skupni čas vsakega izmed teh dveh stanj.
Naslednje kolone so namenjene klicem. V kolonah ″CallInStart″, ″CallOutStart″ in ″RingTimeStart″ sta prikazana datum in čas vsakega izmed teh dogodkov. ″CallsInTime″, ″CallsOutTime″ in ″RingingTime″ kolone pa so namenjene prikazu skupnega časa vsakega izmed teh dogodkov. Na koncu so še kolone ″CallsIn″ v kateri je prikazana vsota vseh dohodnih klicev, ″CallsOut″, kjer so sešteti vsi odhodni klici, ″MissedCalls″, kjer so sešteti vsi neodgovorjeni klici in na koncu ″RingExcess″ kolona, ki nam pove, koliko klicev (tako odgovorjenih kot neodgovorjenih) je zvonilo več kot 10 sekund. Slednja je pomembna predvsem pri dispečerjih, kjer je potreben hiter odziv, saj je od tega lahko odvisno življenje.
V podatkovni mreži na tem zavihku se zgoraj naštete stvari beležijo le dokler je agent prijavljen. Zavihek ″AgentStatus″ je prikazan na slikah 4.4, 4.5 in 4.6.



Slika 4.4: Trenutno stanje agentov 1. del



Slika 4.5: Trenutno stanje agentov 2.del

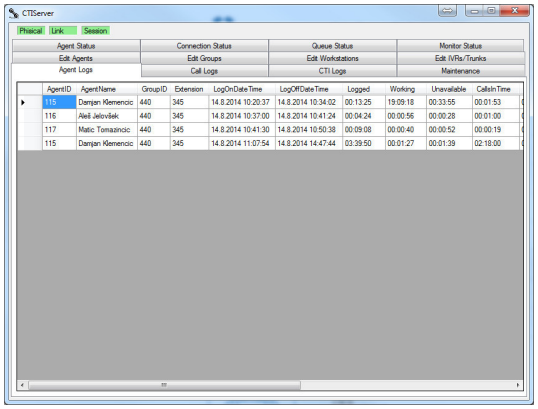


Slika 4.6: Trenutno stanje agentov 3. del

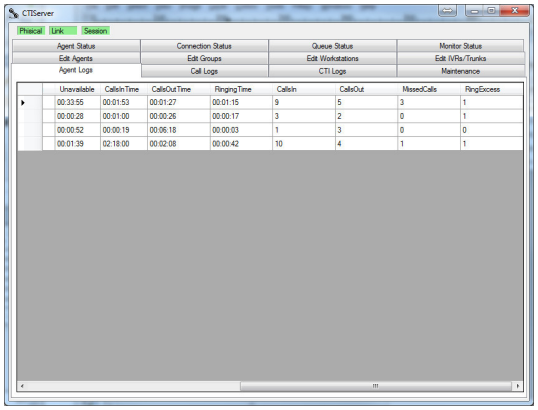
4.2.3 Dnevnik agentskih aktivnosti

V dnevniku agentskih aktivnosti je prikazana zgodovina odjavljenih agentov za obdobje 24 ur. V trenutku, ko se agent odjavi iz sistema, se po-

datki iz podatkovne mreže v zavihku ″AgentStatus″ prepišejo v podatkovno mrežo v tem zavihku. Hkrati pa se vsa polja v podatkovni mreži ″AgentStatus″ postavijo na privzete vrednosti. Prikaz kolon je podoben kot v zavihku ″AgentStatus″. Razlika je v tem, da tukaj ni prikazanih trenutnih stanj kot ″AgentStatus″ in ″TelephonyStatus″. Dodatno je tukaj še kolona ″LogOffDateTime″, ki pove, kdaj se je agent odjavil. Zavihek ″AgentLogs″ je prikazan na slikah 4.7 in 4.8.



Slika 4.7: Dnevnik agentskih aktivnosti 1. del



Slika 4.8: Dnevnik agentskih aktivnosti 2. del

4.3 Klici

4.3.1 Prikaz trenutnih povezav

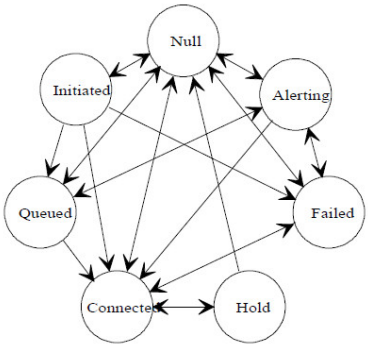
Pod tem zavihkom lahko spremljamo trenutne povezave ter njihovo stanje. Vidimo lahko ″LocalDeviceID″, ki nam pove, katere lokalne številke so trenutno aktivne. ″OtherDeviceID″ nam pove, s katerim telefonom je trenutno vzpostavljena povezava. ″ConnectionID″ je identifikacijska številka povezave. Te številke določa komunikacijski strežnik in gredo od 0 do 65535. Številka se z vsako novo povezavo poveča za 1. ″LocalEvent″ predstavlja dogodek na agentovem telefonu, medtem ko ″OtherEvent″ predstavlja dogodek na nasprotnem telefonu. Možni dogodki so:

- 1 - `conferencedEvent`, ki označuje, da je bila vzpostavljena konferenca in da nobena naprava ni bila odstranjena iz nastalega klica,
- 2 - `connectionClearedEvent`, ki označuje, da je posamezna naprava prekinila klic,
- 3 - `deliveredEvent`, ki označuje, da je bil klic predstavljen napravi z zvonenjem,
- 4 - `divertedEvent`, ki označuje, da je bil klic preusmerjen na drugo napravo in ni več prisoten na napravi,
- 5 - `establishedEvent`, ki označuje, da je bil klic odgovorjen na napravi,
- 6 - `failedEvent`, ki označuje, da klic ne more biti dokončan ali, da je povezava prešla v stanje neuspešno,
- 7 - `heldEvent`, ki označuje, da je bil klic dan na čakanje,
- 8 - `networkReachedEvent`, ki označuje, da je klic prešel skozi meje preklopne poddomene v drugo omrežje,
- 9 - `originatedEvent`, ki označuje, da klic poizkuša iz naprave (nakazuje, da je vnašanje za klic končano in da je bil zahtevan klic),

- 10 - `queuedEvent`, ki označuje, da je klic v vrsti,
- 11 - `retrievedEvent`, ki označuje, da je klic, ki je bil v vrsti, pridobljen nazaj,
- 12 - `serviceInitiatedEvent`, ki označuje, da je telefonska storitev začeta v nadzorovani napravi,
- 13 - `transferedEvent`, ki označuje, da je obstoječi klic prenešen na drugo napravo in, da je naprava, ki je prenesla klic, umaknjena iz klica.

Koloni ″LocalState″ in ″OtherState″ povesta, v kakšnem stanju je trenutno povezava (bodisi v nadzorovani napravi bodisi v nenadzorovani). Možna stanja so:

- 0 - `Neaktivno (StateNull)` - povezava je neaktivna
- 1 - `Začeto (Initiated)` - povezava je začeta
- 2 - `Povezano (Connected)` - povezava je vzpostavljena
- 3 - `Opozarjajoče (Alerting)` - povezava je v stanju zvonenja
- 4 - `Neuspešno (Failed)` - povezava ni uspela
- 5 - `Na čakanju (Hold)` - povezava je na čakanju
- 6 - `V vrsti (Queued)` - povezava je v vrsti



Slika 4.9: Prehodi med stanji [10]

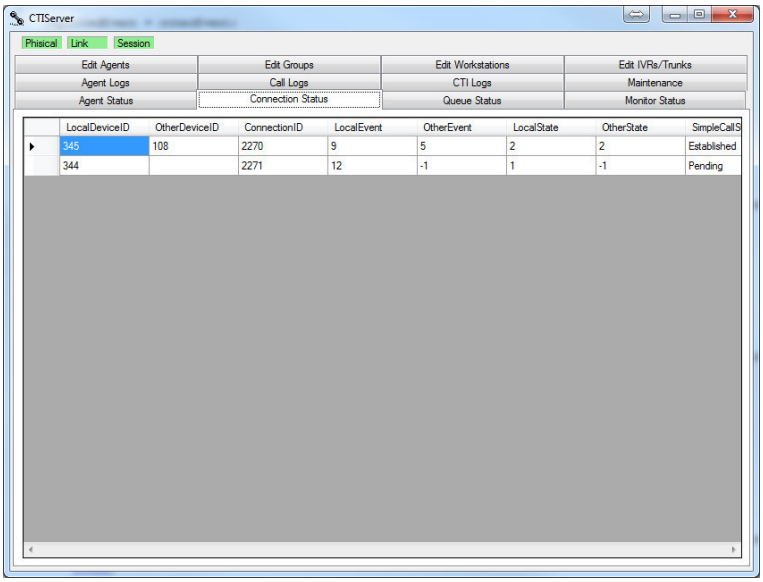
Stanja (predstavljena s krogci), ki so predstavljena na sliki 4.9, sestavljajo niz CSTA [12] stanj povezav. Možnosti prehodov med stanji so predstavljene s puščicami in te so podlaga za zagotavljanje sporočil o dogodkih, ko se takšni prehodi zgodijo.

Stanje lokalne povezave (Local Connection State)	**Stanje druge povezave (Other Connection State)**	**CSTA preprosto stanje klica (CSTA Simple Call Stete)**
Opozarjanje (Alerting)	Povezano (Connected)	Prejeto (Received)
Opozarjanje (Alerting)	Na čakanju (Hold)	Prejeto-Na čakanju (Received-On Hold)
Povezano (Connected)	Opozarjanje (Alerting)	Dostavljeno (Delivered)
Povezano (Connected)	Povezano (Connected)	Vzpostavljeno (Established)
Povezano (Connected)	Neuspešno (Failed)	Neuspešno (Failed)
Povezano (Connected)	Na čakanju (Hold)	Vzpostavljeno-Na čakanju (Established-On Hold)
Povezano (Connected)	Neaktivno (Null)	Zečeto (Originated) / Prekinjeno (Terminated)
Povezano (Connected)	V vrsti (Queued)	V vrsti (Queued)
Neuspešno (Failed)	Neaktivno (Null)	Blokirano (Blocked)
Na čakanju (Hold)	Opozarjanje (Alerting)	Dostavljeno-Zadržano (Delivered-Held)
Na čakanju (Hold)	Povezano (Connected)	Vzpostavljeno-Zadržano (Established-Held)
Na čakanju (Hold)	Neuspešno (Failed)	Neuspešno-Zadržano (Failed-Held)
Na čakanju (Hold)	V vrsti (Queued)	Na čakanju-Zadržano (Queued-Held)
Začeto (Initiated)	Neaktivno (Null)	V pričakovanju (Pending)
Neaktivno (Null)	Neaktivno (Null)	Neaktivno (Null)

Tabela 4.1: CSTA preprosta stanja klicev

Zadnja kolona ″SimpleCallState″ je kombinacija kolon ″LocalState″ in ″OtherState″. V tabeli 4.1 so prikazane vse kombinacije teh stanj.

Zavihek ″Connection Status″ je prikazan na sliki 4.10 prikazuje .



Slika 4.10: Trenutne povezave

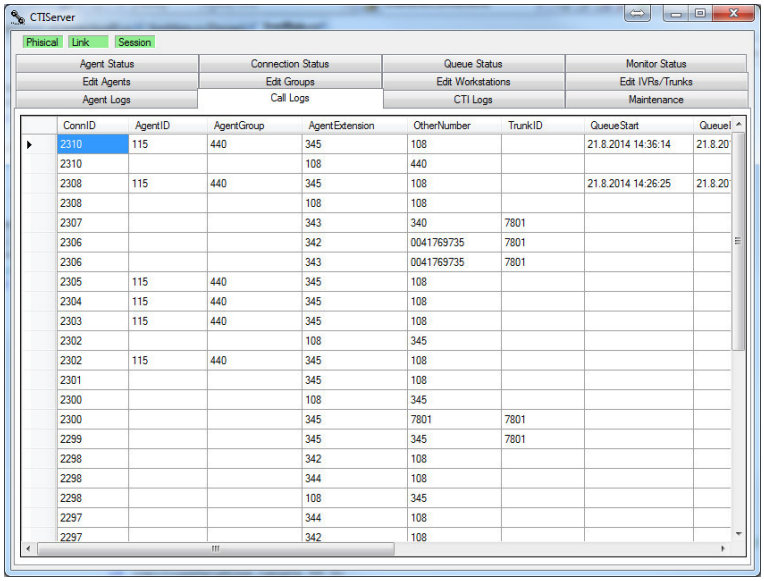
4.3.2 Dnevnik klicev

V zavihku ″AgentLogs″ najdemo podatke o opravljenih klicih za obdobje zadnjih 24 ur. Podatkovna mreža vsebuje naslednje kolone:

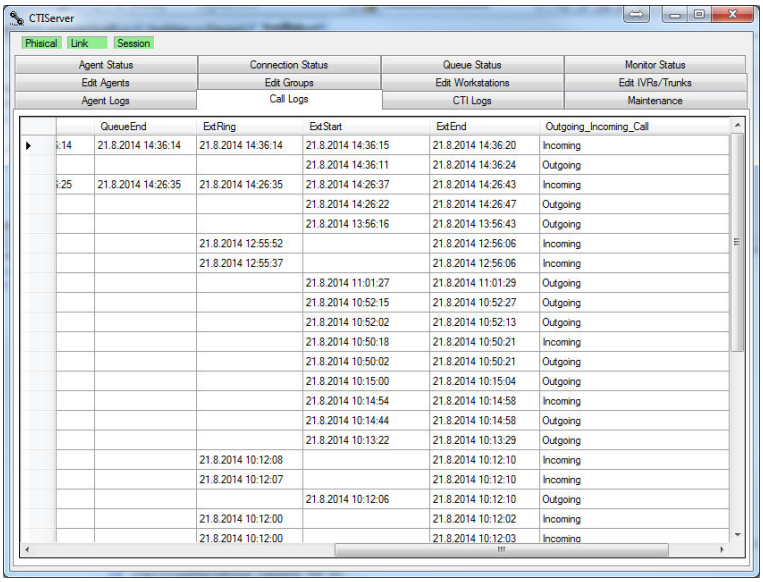
- ″ConnID″ predstavlja zaporedno številko zveze. Z vsako novo zvezo se ta številka poveča za 1.
- ″AgentID″ predstavlja identifikacijsko številko agenta. Vsakemu agentu pripada svoja številka.
- ″AgentGroup″ je identifikacijska številka skupine, kateri pripada agent.
- ″AgentExtension″ nam pove, na katero delovno mesto je agent prijavljen.
- ″OtherNumber″ je podatek o nasprotni telefonski številki (lahko je to lokalna številka znotraj sistema ali pa številka zunaj sistema).

- ″TrunkID″ je podatek o napravi, preko katere smo dosegli zunanjo mrežo.
- ″QueueStart″ hrani datum in čas, ko je klic prispel v vrsto.
- ″QueueEnd″ beleži čas, ko je klic zapustil vrsto.
- ″ExtRing″ predstavlja čas, ko je agentu začel zvoniti telefon. Če je bil klic pred tem v vrsti, sta časa ″QueueEnd″ in ″ExtRing″ enaka.
- ″ExtStart″ beleži čas, ko se je agent odzval na klic.
- ″ExtEnd″ hrani čas, ko je agent končal klic.
- ″Outgoing_Incoming_Call″ pove, v katero smer je šel klic (ali je bil odhoden ali dohoden).

Zavihek ″Call Logs″ je prikazan na slikah 4.11 in 4.12.



Slika 4.11: Dnevnik klicev 1. del



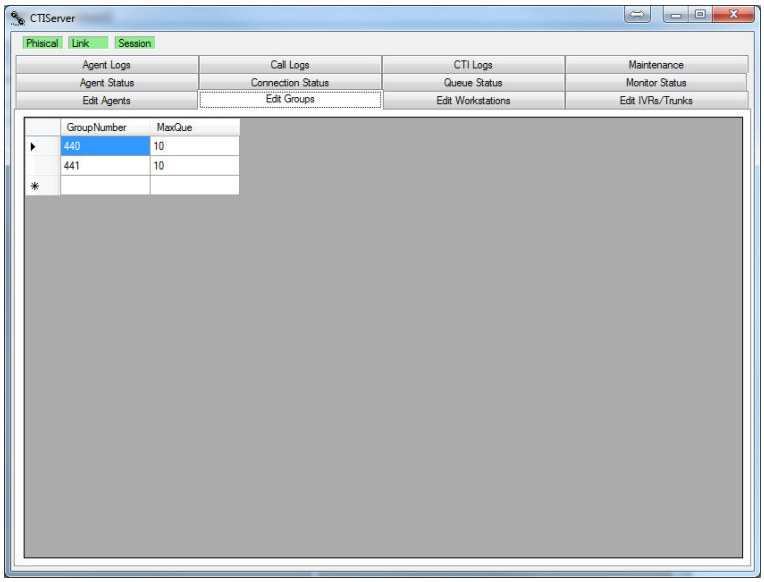
Slika 4.12: Dnevnik klicev 2. del

4.4 Urejanje

Pod urejanje spadajo zavihki ″Edit Groups″, ″Edit Workstations″ in ″Edit IVRs″. Sem bi lahko uvrstili tudi zavihek ″Edit Agents″, vendar smo ga že obdelali v poglavju 4.2, kjer smo zajeli vse, kar se tiče agentov. Vse iz zgoraj omenjenih zavihkihov se mora ujemati s podatki v komunikacijskem strežniku. Ti zavihki so namenjeni različnim vrstam naprav. Ko se aplikacija zažene, pošlje komunikacijskemu strežniku ukaz za nadzor za vsako napravo posebej.

4.4.1 Urejanje skupin

V tem zavihku lahko dodajamo, brišemo ali le spreminjamo nastavitve skupin. Spremeniti je možno številko skupine in največje število klicev v vrsti. Slika 4.13 prikazuje zavihek ″Edit Groups″.



Slika 4.13: Urejanje skupin

4.4.2 Urejanje delovnih postaj

Tukaj lahko dodajamo, brišemo ali le spreminjamo nastavitve za delovne postaje. Najpomembnejši vnos je kolona ″Extension″, ki mora biti nujno pravilno (z obstoječo napravo) zapolnjena. Koloni ″WorkStationApplicationPath″ in ″WorkStationIPAddress″ sta namenjeni dodatnim aplikacijam, ki so naložene na delovnih postajah (npr.: CtiClient). Teh ne bom posebej opisovali, ker za to aplikacijo niso pomembne.

4.4.3 Urejanje interaktivnih glasovnih odzivnikov in naprav za dostop do zunanje mreže

Interaktivni glasovni odzivnik (ang. Interactive Voice Response - IVR) je tehnologija, ki omogoča interakcijo med računalnikom in človekom preko uporabe glasu ali preko tipk na telefonu. IVR se lahko odzove na klic s posnetim ali dinamično generiranim zvokom in tako naprej usmeri uporabnika [8]. V kolono ″IVRID″ je potrebno vpisati vse IVR številke.

Naprava za dostop do zunanje mreže (ang. trunk) je naprava, ki se uporablja za dostop do preklopnih poddomen. Preko te naprave lahko dosežemo zunanjo mrežo in tako opravljamo klice izven našega sistema. To je pravzaprav prenosni kanal med dvema točkama [10]. V kolono ″TrunkID″ je potrebno vpisati vse številke naprav, preko katerih dosegamo zunanjo mrežo.

4.5 Stanje naprav

V zavihku ″Monitor Status″ lahko vidimo, katere naprave so nadzorovane in kakšen je njihov namen. Nadzorujejo se vse naprave ki so vpisane v podatkovnih mrežah pod zavihki ″Edit Groups″, ″Edit Workstations″ in ″Edit IVRs″. Številke nadzorovanih naprav so v koloni ″Device″, tip naprave pa je zapisan v koloni ″Id_typ″. Možni tipi naprav so:

- 1 - `IVR` (ang. Interactive voice response) tehnologija, ki omogoča interakcijo med računalnikom in človekom preko glasu ali tipk na telefonu.
- 2 - `Queue` ali `ACD grouop` (ang. Automatic Call Distributor group) je mehanizem, ki s pomočjo funkcije prklapljanja (ang. Switching Function) daje v vrsto in razporeja tako klice kot tudi naprave, na katere ta mehanizem razporeja klice.
- 3 - `WorkStation` je delovna postaja. Ponavadi je to telefon in računalnik, lahko pa tudi samo telefon.
- 4 - `Trunk` je naprava, preko katere lahko dosežemo zunanjo mrežo.

4.6 Dnevnik komunikacije

Na ta zavihek je postavljen kontrolnik ctlCC, preko katerega poteka vsa komunikacija med komunikacijskim strežnikom in aplikacijo. Kontrolnik vsebuje tri tekstovna polja, ki so namenjena prikazu originalnih podatkov na

podatkovnem (v smeri proti komunikacijskemu strežniku in obratno) in sejnem sloju modela ISO OSI 7 (glej tabelo 2.2). Kontrolnik vsebuje tudi serijski port, preko katerega poteka komunikacija. Ko kontrolnik prejme podatek iz komunikacijskega strežnika, ga obdela in posreduje izluščene podatke preko dogodka `CC1_eChangeMonitor` glavni formi. Glavna forma ugotovi, za kakšno vrsto podatka gre in prikaže ustrezen odziv.

Poglavje 5

Sklepne ugotovitve

Do sedaj smo aplikacijo postavili v dveh klicnih centrih in sicer v Celju in Ljubljani. Ugotovili smo, da so uporabniki z njo zelo zadovoljni in na njo nimajo pripomb. Razvoj aplikacije od ideje do izvedbe je trajal približno eno leto. Od tega smo testiranju in odpravljanju napak namenili približno en mesec. Naredili smo vse, kar smo si na začetki projekta zastavili. Aplikacija teče na strežniku, ki se nahaja v istem prostoru kot komunikacijski strežnik, saj mora biti zaradi delovanja aplikacije povezan s centralo. Za analizo podatkov, ki jih pridobi naša aplikacija, stranke uporabljajo dodatni program `CCManager` (ang. Call Center Manager), ki ni del te aplikacije.

Izdelava aplikacije ni povzročala večjih težav, ker smo pred pričetkom izdelave dobro preučili dokumentacijo, predvsem protokol CSTA. Nekaj težav se je vseeno pojavilo, saj v določenih delih dokumentacija ni najbolj natančna. V aplikaciji smo zajeli vse želje potencialnih strank, dodali še kakšno dodatno funkcionalnost in s tem uspešno izpolnili na začetku projekta zastavljene cilje. Grafični vmesnik je preprost in dokaj pregleden, kar omogoča hiter pregled trenutnega stanja klicnega centra.
Aplikacija ima kar nekaj možnosti za nadalnji razvoj. V prihodnosti bi lahko naredili skoraj celotno funkcionalnost telefona preko računalnika. V prvi fazi nadgradnje bi bila dovolj že vzpostavitev klicanja in sprejemanja klicev. Na delovne postaje bi naložili programsko opremo, ki bi pošiljala ukaze v CtiSer-

ver, ta pa bi preko protokola CSTA pošiljal ukaze naprej komunikacijskemu strežniku.

Slike

Tabele

Literatura

[1] Lars Powers, Mike Snell. Microsoft Visual Studio unleashed, Sams Publishing, 2008.

[2] Jesse Lierty, Donald Xie. Programming C# 3.0, O'Reilly, 2008.

[3] (2014) History RS232. Dostopno na:
http://www.lookrs232.com/rs232/

[4] (2014) The RS232 standard. Dostopno na:
http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html#anchor1155222

[5] IBM. General Information - Binary Synchronous Communications Third Edition, 1970

[6] (2014) Microsoft SQL Server. Dostopno na:
http://en.wikipedia.org/wiki/Microsoft_SQL_Server#SQL_Server_2008_R2

[7] William A. Yarberry, Jr.. Computer Telephony Integration Second Edition, CRC Press LLC, 2003.

[8] (2014) Interactive voice response. Dostopno na:
http://en.wikipedia.org/wiki/Interactive_voice_response

[9] (2014) Ecma International. Dostopno na:
http://www.ecma-international.org

[10] Standard ECMA-217 Service for Computer Supported Telecommunications Applications (CSTA) Phase II, december 1994

[11] Standard ECMA-218 Protocol for Computer Supported Telecommunications Applications (CSTA) Phase II, december 1994

[12] (2014) CSTA protocol. Dostopno na strani: http://wiki.unify.com/wiki/HiPath_3000_open_interfaces

[13] (2014) HiPath 3000. Dostopno na strani: http://wiki.unify.com/wiki/HiPath_3000